```
import sys

# rounds to integer, 0.2->0, 0.8->1
def round_int(x):
    return ((10*(x + 0.5)) // 10)

from pylab import *

# create time vector, with sampling frequency 10,000, with even number of samples
t = arange(0, 1.0001, 1.0/10000)

# create the signal as a equally weighted sum of harmonics at with f = 100, 600, 700 and 800
s = sin(1*2*pi*100*t) + sin(6*2*pi*100*t) + sin(7*2*pi*100*t) + sin(8*2*pi*100*t)
# normalize the signal and plot
s = s / max(abs(s))
plot(s[300:600])
xlabel('time'); ylabel('original signal');
show()

# shift to positive, re-normalize and scale up by 2^8-1
sa = ((s + 1)/2)*(2**8-1)
# quantize, remove decimals, and plot signal and error
sq = round_int(sa)
plot(sq[300:600])
plot(sq[300:600]-sa[300:600], 'red')
xlabel('time'); ylabel('quantised signal / error');
show()

ft = fft(s)/len(s)
plot(20*log10(abs(ft)))
xlabel('frequency'); ylabel('signal FFT [dB]');
show()

ft = fft(sq)/len(sq)
plot(20*log10(abs(ft)))
xlabel('frequency'); ylabel('quantised signal FFT [dB]');
show()

# FIR filter coefficients as given by http://www.arc.id.au/FilterDesign.html
b=array([
0.000685,
0.002825,
0.006690,
0.012355,
0.019402,
0.026914,
0.033645,
0.038324,
0.040000,
0.038324,
0.033645,
0.026914,
0.019402,
0.012355,
0.006690,
0.002825,
0.000685])
plot(b)
xlabel('n'); ylabel('coefficients b[n]');
show()

# normalize coefficients for a more accurate quantisation
ba = b / max(b) # coefficients happened to be all positive
# scale up by 2^8-1
bas = ba*(2**8-1)
```

```
66  # quantize, remove decimals, and plot
67  bq = round_int(bas)
68  plot(bq)
69  plot(bq-bas, 'red')
70  xlabel('n'); ylabel('quantised coefficients bq[n] / error');
71  show()
72
73  # initialize registers and outputs to zero (float, integer)
74  r=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
75  ri=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
76  y=[0] * len(t)
77  yi=[0] * len(t)
78
79  for n in xrange(0, len(t)):
80      y[n] = s[n]*b[0] + r[0]*b[1] + r[1]*b[2] + r[2]*b[3] + r[3]*b[4] + r[4]*b[5] +
        r[5]*b[6] + r[6]*b[7] + r[7]*b[8] + r[8]*b[9] + r[9]*b[10] + r[10]*b[11] + r[11
        ]*b[12] + r[12]*b[13] + r[13]*b[14] + r[14]*b[15] + r[15]*b[16]
81      r[14] = r[13]
82      r[13] = r[12]
83      r[12] = r[11]
84      r[11] = r[10]
85      r[10] = r[9]
86      r[9] = r[8]
87      r[8] = r[7]
88      r[7] = r[6]
89      r[6] = r[5]
90      r[5] = r[4]
91      r[4] = r[3]
92      r[3] = r[2]
93      r[2] = r[1]
94      r[1] = r[0]
95      r[0] = s[n]
96
97      yi[n] = sq[n]*bq[0] + ri[0]*bq[1] + ri[1]*bq[2] + ri[2]*bq[3] + ri[3]*bq[4] +
        ri[4]*bq[5] + ri[5]*bq[6] + ri[6]*bq[7] + ri[7]*bq[8] + ri[8]*bq[9] + ri[9]*bq[
        10] + ri[10]*bq[11] + ri[11]*bq[12] + ri[12]*bq[13] + ri[13]*bq[14] + ri[14]*bq
        [15] + ri[15]*bq[16]
98      ri[14] = ri[13]
99      ri[13] = ri[12]
100     ri[12] = ri[11]
101     ri[11] = ri[10]
102     ri[10] = ri[9]
103     ri[9] = ri[8]
104     ri[8] = ri[7]
105     ri[7] = ri[6]
106     ri[6] = ri[5]
107     ri[5] = ri[4]
108     ri[4] = ri[3]
109     ri[3] = ri[2]
110     ri[2] = ri[1]
111     ri[1] = ri[0]
112     ri[0] = sq[n]
113
114 # plot outputs (float, integer)
115 plot(y[300:600])
116 xlabel('time'); ylabel('filtered signal');
117 show()
118
119 plot(yi[300:600])
120 xlabel('time'); ylabel('q-filtered signal');
121 show()
122
123 # post process the q-filtered signal
124 yi = array(yi)
125 # scale back down
126 yis = (yi*max(b)*2)/((2**8-1)**2)
```

```
# shift down
middle = (max(yis) + min(yis[400:]))/2
yq = yis-middle
plot(yq[300:600])
plot(yq[300:600]-y[300:600], 'red')
xlabel('time'); ylabel('processed q-filtered signal / error');
show()

# plot FFT of filtered signal (float, integer)
ft = fft(y)/len(y)
plot(20*log10(abs(ft)))
xlabel('frequency'); ylabel('filtered signal FFT [dB]');
show()

ft = fft(yq)/len(yq)
plot(20*log10(abs(ft)))
xlabel('frequency'); ylabel('q-filtered signal FFT [dB]');
show()

sys.exit("end")
```